Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática
Escuela Profesional de Ingeniería de Estadística e Informática

Definiciones : Variables, funciones y Restricciones Ejercicios

Curso:

MÉTODOS DE OPTIMIZACIÓN

Presentado por:

JEAN CARLOS WILLIAM HUANCOILLO ROJAS

Docente:

FRED TORRES CRUZ

Puno – Perú

2025

1. El precio de una vivienda (P) depende linealmente del área construida (A) y puede expresarse como P = mA + b, donde m es el costo por metro cuadrado y b representa costos fijos.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_precio_vivienda(m, b, area_min, area_max):
    """
    Calcula y grafica el precio de una vivienda en función del área construida.

    Parámetros:
    m (float): Costo por metro cuadrado.
    b (float): Costos fijos.
    area_min (int): Área mínima.
    area_max (int): Área máxima.
    """
    # Crear un rango de áreas
    areas = np.linspace(area_min, area_max, 100)

    # Calcular el precio para cada área
    precios = m * areas + b

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(areas, precios, label=f"P = {m}A + {b}", color="blue")
    plt.title("Precio de la Vivienda en función del Área Construida")
    plt.xlabel("Área Construida (m²)")
    plt.ylabel("Precio de la Vivienda")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Precio de Vivienda: P = mA + b")
m = float(input("Ingrese el costo por metro cuadrado (m): "))
b = float(input("Ingrese los costos fijos (b): "))
area_min = int(input("Ingrese el área mínima (m²): "))
area_max = int(input("Ingrese el área máxima (m²): "))
area_especifica = float(input("Ingrese un área específica para calcular su precio: "))

# Resolver la función para el área específica
precio_especifico = m * area_especifica + b
print(f"El precio de la vivienda para un área de {area_especifica} m² es: {precio_especifico}")

# Llamar a la función para graficar
calcular_precio_vivienda(m, b, area_min, area_max)
```

Este código calcula y gráfica el precio de una vivienda en función del área construida usando la ecuación P = mA + b**. Además, permite resolver la función para un área específica ingresada por el usuario, mostrando el precio resultante en la consola. Combina visualización y cálculo directo en un solo programa.**

2. La ganancia mensual (G) de un modelo depende linealmente del número de predicciones realizadas (N) como G = cN + b, donde c es la ganancia por predicción y b son ingresos fijos.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_ganancia_mensual(c, b, n_min, n_max):
```

```
    """
    Calcula y grafica la ganancia mensual en función del número de predicciones realizadas.

    Parámetros:
    c (float): Ganancia por predicción.
    b (float): Ingresos fijos.
    n_min (int): Número mínimo de predicciones.
    n_max (int): Número máximo de predicciones.
    """
    # Crear un rango de predicciones
    predicciones = np.linspace(n_min, n_max, 100)

    # Calcular la ganancia para cada número de predicciones
    ganancias = c * predicciones + b

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(predicciones, ganancias, label=f"G = {c}N + {b}", color="green")
    plt.title("Ganancia Mensual en función del Número de Predicciones")
    plt.xlabel("Número de Predicciones (N)")
    plt.ylabel("Ganancia Mensual (G)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Ganancia Mensual: G = cN + b")
c = float(input("Ingrese la ganancia por predicción (c): "))
b = float(input("Ingrese los ingresos fijos (b): "))
n_min = int(input("Ingrese el número mínimo de predicciones: "))
n_max = int(input("Ingrese el número máximo de predicciones: "))
n_especifico = float(input("Ingrese un número específico de predicciones para calcular la ganancia: "))

# Resolver la función para un número específico de predicciones
ganancia_especifica = c * n_especifico + b
print(f"La ganancia mensual para {n_especifico} predicciones es: {ganancia_especifica}")

# Llamar a la función para graficar
calcular_ganancia_mensual(c, b, n_min, n_max)
```

El código está adaptado para calcular y graficar la ganancia mensual G = cN + b basado en el número de predicciones realizadas. También permite calcular la ganancia para un número específico de predicciones ingresadas por el usuario

3. El tiempo total de procesamiento (T) en un algoritmo depende linealmente del tamaño de los datos (D), expresado como T = kD + c, donde k es el tiempo por unidad de datos y c es un tiempo constante de configuración.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_tiempo_procesamiento(k, c, d_min, d_max):
    """
    Calcula y grafica el tiempo total de procesamiento en función del tamaño de los datos.

    Parámetros:
```

```
    k (float): Tiempo por unidad de datos.
    c (float): Tiempo constante de configuración.
    d_min (int): Tamaño mínimo de los datos.
    d_max (int): Tamaño máximo de los datos.
    """
    # Crear un rango de tamaños de datos
    datos = np.linspace(d_min, d_max, 100)

    # Calcular el tiempo para cada tamaño de datos
    tiempos = k * datos + c

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(datos, tiempos, label=f"T = {k}D + {c}", color="purple")
    plt.title("Tiempo Total de Procesamiento en función del Tamaño de los Datos")
    plt.xlabel("Tamaño de los Datos (D)")
    plt.ylabel("Tiempo Total de Procesamiento (T)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Tiempo Total de Procesamiento: T = kD + c")
k = float(input("Ingrese el tiempo por unidad de datos (k): "))
c = float(input("Ingrese el tiempo constante de configuración (c): "))
d_min = int(input("Ingrese el tamaño mínimo de los datos: "))
d_max = int(input("Ingrese el tamaño máximo de los datos: "))
d_especifico = float(input("Ingrese un tamaño específico de datos para calcular el tiempo total: "))

# Resolver la función para un tamaño específico de datos
tiempo_especifico = k * d_especifico + c
print(f"El tiempo total de procesamiento para un tamaño de datos de {d_especifico} es: {tiempo_especifico}")

# Llamar a la función para graficar
calcular_tiempo_procesamiento(k, c, d_min, d_max)
```

Este código calcula y gráfica el tiempo total de procesamiento (T) en función del tamaño de los datos (D) usando la fórmula T = kD + c. También permite calcular el tiempo para un tamaño específico de datos.

4. El costo total (C) para almacenar datos depende linealmente de la cantidad de datos almacenados (D), según C = pD + f, donde p es el costo por gigabyte y f son tarifas fijas.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_costo_almacenamiento(p, f, d_min, d_max):
    """
    Calcula y grafica el costo total de almacenamiento en función de la cantidad de datos almacenados.

    Parámetros:
    p (float): Costo por gigabyte.
    f (float): Tarifas fijas.
    d_min (int): Cantidad mínima de datos almacenados.
    d_max (int): Cantidad máxima de datos almacenados.
    """
    # Crear un rango de tamaños de datos
    datos = np.linspace(d_min, d_max, 100)
```

```
    # Calcular el costo para cada tamaño de datos
    costos = p * datos + f

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(datos, costos, label=f"C = {p}D + {f}", color="blue")
    plt.title("Costo Total de Almacenamiento en función de la Cantidad de Datos")
    plt.xlabel("Cantidad de Datos Almacenados (D)")
    plt.ylabel("Costo Total de Almacenamiento (C)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Costo Total de Almacenamiento: C = pD + f")
p = float(input("Ingrese el costo por gigabyte (p): "))
f = float(input("Ingrese las tarifas fijas (f): "))
d_min = int(input("Ingrese la cantidad mínima de datos almacenados: "))
d_max = int(input("Ingrese la cantidad máxima de datos almacenados: "))
d_especifico = float(input("Ingrese una cantidad específica de datos para calcular el costo total: "))

# Resolver la función para una cantidad específica de datos
costo_especifico = p * d_especifico + f
print(f"El costo total para almacenar {d_especifico} gigabytes de datos es: {costo_especifico}")

# Llamar a la función para graficar
calcular_costo_almacenamiento(p, f, d_min, d_max)
```

Este código calcula y gráfica el costo total de almacenamiento (C) en función de la cantidad de datos almacenados (D) usando la fórmula C = pD + f. D+F. También permite calcular el costo para una cantidad específica de datos.

5. La medición calibrada (M) de un sensor depende linealmente de la medición en crudo (R) como M = aR + b, donde a es el factor de ajuste y b es un desplazamiento constante.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_medicion_calibrada(a, b, r_min, r_max):
    """
    Calcula y grafica la medición calibrada en función de la medición en crudo.

    Parámetros:
    a (float): Factor de ajuste.
    b (float): Desplazamiento constante.
    r_min (int): Valor mínimo de la medición en crudo.
    r_max (int): Valor máximo de la medición en crudo.
    """
    # Crear un rango de mediciones en crudo
    r = np.linspace(r_min, r_max, 100)

    # Calcular la medición calibrada para cada medición en crudo
    m = a * r + b

    # Graficar la función
    plt.figure(figsize=(8, 6))
```

```
    plt.plot(r, m, label=f"M = {a}R + {b}", color="red")
    plt.title("Medición Calibrada en función de la Medición en Crudo")
    plt.xlabel("Medición en Crudo (R)")
    plt.ylabel("Medición Calibrada (M)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Medición Calibrada: M = aR + b")
a = float(input("Ingrese el factor de ajuste (a): "))
b = float(input("Ingrese el desplazamiento constante (b): "))
r_min = int(input("Ingrese la medición mínima en crudo (R): "))
r_max = int(input("Ingrese la medición máxima en crudo (R): "))
r_especifico = float(input("Ingrese una medición específica en crudo para calcular su medición calibrada: "))

# Resolver la función para la medición en crudo específica
m_especifico = a * r_especifico + b
print(f"La medición calibrada para una medición en crudo de {r_especifico} es: {m_especifico}")

# Llamar a la función para graficar
calcular_medicion_calibrada(a, b, r_min, r_max)
```

El código calcula y gráfica la medición calibrada en función de la medición en crudo utilizando la ecuación M = aR + b, permitiendo al usuario ingresar parámetros y obtener resultados para un valor específico de R.

6. El tiempo de respuesta promedio (T) de un servidor depende linealmente del número de solicitudes simultáneas (S) como T = mS +b, donde m es el tiempo incremental por solicitud y b es el tiempo base.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_tiempo_respuesta(m, b, s_min, s_max):
    """
    Calcula y grafica el tiempo de respuesta promedio en función del número de solicitudes simultáneas.

    Parámetros:
    m (float): Tiempo incremental por solicitud.
    b (float): Tiempo base.
    s_min (int): Número mínimo de solicitudes simultáneas.
    s_max (int): Número máximo de solicitudes simultáneas.
    """
    # Crear un rango de solicitudes simultáneas
    solicitudes = np.linspace(s_min, s_max, 100)

    # Calcular el tiempo de respuesta para cada número de solicitudes
    tiempos = m * solicitudes + b

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(solicitudes, tiempos, label=f"T = {m}S + {b}", color="orange")
    plt.title("Tiempo de Respuesta Promedio en función del Número de Solicitudes Simultáneas")
    plt.xlabel("Número de Solicitudes Simultáneas (S)")
    plt.ylabel("Tiempo de Respuesta Promedio (T)")
    plt.grid(True)
    plt.legend()
    plt.show()
```

```
    # Pedir al usuario los valores necesarios
    print("Modelo de Tiempo de Respuesta Promedio: T = mS + b")
    m = float(input("Ingrese el tiempo incremental por solicitud (m): "))
    b = float(input("Ingrese el tiempo base (b): "))
    s_min = int(input("Ingrese el número mínimo de solicitudes simultáneas (S): "))
    s_max = int(input("Ingrese el número máximo de solicitudes simultáneas (S): "))
    s_especifico = float(input("Ingrese un número específico de solicitudes para calcular el tiempo de respuesta: "))

    # Resolver la función para un número específico de solicitudes
    tiempo_especifico = m * s_especifico + b
    print(f"El tiempo de respuesta para {s_especifico} solicitudes simultáneas es: {tiempo_especifico}")

    # Llamar a la función para graficar
    calcular_tiempo_respuesta(m, b, s_min, s_max)
```

El código calcula y gráfica el tiempo de respuesta promedio de un servidor en función del número de solicitudes simultáneas utilizando la ecuación. T = mS +b, permitiendo al usuario ingresar parámetros y obtener resultados para un valor específico de S.

7. Los ingresos (I) de una plataforma dependen linealmente del número de suscriptores (S) como I = pS + b, donde p es el ingreso promedio por suscriptor y b son ingresos adicionales.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_ingresos(p, b, s_min, s_max):
    """
    Calcula y grafica los ingresos de una plataforma en función del número de suscriptores.

    Parámetros:
    p (float): Ingreso promedio por suscriptor.
    b (float): Ingresos adicionales.
    s_min (int): Número mínimo de suscriptores.
    s_max (int): Número máximo de suscriptores.
    """
    # Crear un rango de suscriptores
    suscriptores = np.linspace(s_min, s_max, 100)

    # Calcular los ingresos para cada número de suscriptores
    ingresos = p * suscriptores + b

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(suscriptores, ingresos, label=f"I = {p}S + {b}", color="purple")
    plt.title("Ingresos en función del Número de Suscriptores")
    plt.xlabel("Número de Suscriptores (S)")
    plt.ylabel("Ingresos (I)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Ingresos: I = pS + b")
p = float(input("Ingrese el ingreso promedio por suscriptor (p): "))
b = float(input("Ingrese los ingresos adicionales (b): "))
s_min = int(input("Ingrese el número mínimo de suscriptores (S): "))
```

```
s_max = int(input("Ingrese el número máximo de suscriptores (S): "))
s_especifico = float(input("Ingrese un número específico de suscriptores para calcular los ingresos: "))

# Resolver la función para un número específico de suscriptores
ingreso_especifico = p * s_especifico + b
print(f"Los ingresos para {s_especifico} suscriptores son: {ingreso_especifico}")

# Llamar a la función para graficar
calcular_ingresos(p, b, s_min, s_max)
```

El código calcula y gráfica los ingresos de una plataforma en función del número de suscriptores utilizando la ecuación I = pS + b.

8. La energía consumida (E) depende linealmente del número de operaciones realizadas (O) como E = kO + b, donde k es la energía consumida por operación y b es la energía base para encender el sistema.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_energia_consumida(k, b, o_min, o_max):
    """
    Calcula y grafica la energía consumida en función del número de operaciones realizadas.

    Parámetros:
    k (float): Energía consumida por operación.
    b (float): Energía base para encender el sistema.
    o_min (int): Número mínimo de operaciones.
    o_max (int): Número máximo de operaciones.
    """
    # Crear un rango de operaciones
    operaciones = np.linspace(o_min, o_max, 100)

    # Calcular la energía consumida para cada número de operaciones
    energia = k * operaciones + b

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(operaciones, energia, label=f"E = {k}O + {b}", color="red")
    plt.title("Energía Consumida en función del Número de Operaciones")
    plt.xlabel("Número de Operaciones (O)")
    plt.ylabel("Energía Consumida (E)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Energía Consumida: E = kO + b")
k = float(input("Ingrese la energía consumida por operación (k): "))
b = float(input("Ingrese la energía base para encender el sistema (b): "))
o_min = int(input("Ingrese el número mínimo de operaciones (O): "))
o_max = int(input("Ingrese el número máximo de operaciones (O): "))
o_especifico = float(input("Ingrese un número específico de operaciones para calcular la energía consumida: "))

# Resolver la función para un número específico de operaciones
energia_especifica = k * o_especifico + b
print(f"La energía consumida para {o_especifico} operaciones es: {energia_especifica}")
```

```
# Llamar a la función para graficar
calcular_energia_consumida(k, b, o_min, o_max)
```

El código calcula y gráfica la energía consumida en función del número de operaciones realizadas utilizando la ecuación. E = kO + b.

9. El número de likes (L) en una publicación depende linealmente del número de seguidores (F) como L = mF + b, donde m es la proporción promedio de interacción y b es un nivel base de likes.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_likes(m, b, f_min, f_max):
    """
    Calcula y grafica el número de likes en función del número de seguidores.

    Parámetros:
    m (float): Proporción promedio de interacción.
    b (float): Nivel base de likes.
    f_min (int): Número mínimo de seguidores.
    f_max (int): Número máximo de seguidores.
    """
    # Crear un rango de seguidores
    seguidores = np.linspace(f_min, f_max, 100)

    # Calcular el número de likes para cada número de seguidores
    likes = m * seguidores + b

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(seguidores, likes, label=f"L = {m}F + {b}", color="orange")
    plt.title("Número de Likes en función del Número de Seguidores")
    plt.xlabel("Número de Seguidores (F)")
    plt.ylabel("Número de Likes (L)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Likes: L = mF + b")
m = float(input("Ingrese la proporción promedio de interacción (m): "))
b = float(input("Ingrese el nivel base de likes (b): "))
f_min = int(input("Ingrese el número mínimo de seguidores (F): "))
f_max = int(input("Ingrese el número máximo de seguidores (F): "))
f_especifico = float(input("Ingrese un número específico de seguidores para calcular los likes: "))

# Resolver la función para un número específico de seguidores
likes_especifico = m * f_especifico + b
print(f"El número de likes para {f_especifico} seguidores es: {likes_especifico}")

# Llamar a la función para graficar
calcular_likes(m, b, f_min, f_max)
```

El código calcula y gráfica el número de me gusta en función del número de seguidores utilizando la ecuación L = mF +b.

10. El costo total (C) para entrenar un modelo de machine learning depende linealmente del número de iteraciones (I) como C = pI + c, donde p es el costo por iteración y c son costos iniciales.

```
import numpy as np
import matplotlib.pyplot as plt

def calcular_costo_total(p, c, i_min, i_max):
    """
    Calcula y grafica el costo total para entrenar un modelo de machine learning
    en función del número de iteraciones.

    Parámetros:
    p (float): Costo por iteración.
    c (float): Costos iniciales.
    i_min (int): Número mínimo de iteraciones.
    i_max (int): Número máximo de iteraciones.
    """
    # Crear un rango de iteraciones
    iteraciones = np.linspace(i_min, i_max, 100)

    # Calcular el costo total para cada número de iteraciones
    costo_total = p * iteraciones + c

    # Graficar la función
    plt.figure(figsize=(8, 6))
    plt.plot(iteraciones, costo_total, label=f"C = {p}I + {c}", color="purple")
    plt.title("Costo Total en función del Número de Iteraciones")
    plt.xlabel("Número de Iteraciones (I)")
    plt.ylabel("Costo Total (C)")
    plt.grid(True)
    plt.legend()
    plt.show()

# Pedir al usuario los valores necesarios
print("Modelo de Costo Total: C = pI + c")
p = float(input("Ingrese el costo por iteración (p): "))
c = float(input("Ingrese los costos iniciales (c): "))
i_min = int(input("Ingrese el número mínimo de iteraciones (I): "))
i_max = int(input("Ingrese el número máximo de iteraciones (I): "))
i_especifico = float(input("Ingrese un número específico de iteraciones para calcular el costo: "))

# Resolver la función para un número específico de iteraciones
costo_especifico = p * i_especifico + c
print(f"El costo total para {i_especifico} iteraciones es: {costo_especifico}")

# Llamar a la función para graficar
calcular_costo_total(p, c, i_min, i_max)
```

El código calcula y gráfica el costo total para entrenar un modelo de aprendizaje automático en función del número de iteraciones utilizando la ecuación, C = pI + c.